IN THE CLAIMS:

Please cancel claim 21 and amend the remaining claims as follows.

1. (Currently amended) A memory entry, storing a ~~A~~ trace having a multiple-entry, single exit architecture.

2. (Currently amended) The memory entry ~~The trace~~ of claim 1, wherein the trace is a complex trace having multiple independent prefixes and a common, shared suffix.

3. (Currently amended) The memory entry ~~trace~~ of claim 1, wherein the entry ~~trace~~ is indexed by an address of a terminal instruction therein.

4. (Previously presented) A front-end system for a processor, comprising:

an instruction cache system;

an extended block cache system, comprising:

a fill unit provided in communication with the instruction cache system,

a block cache,

a block predictor to store masks associated with complex blocks, the masks distinguishing block prefixes from each other; and

a selector coupled to the output of the instruction cache system and to an output of the block cache.

5. (Original) The front-end system of claim 4, wherein the extended block cache system further comprises a block predictor provided in communication with the fill unit and the block cache.

6. (Previously presented) The front-end system of claim 4, wherein the block cache is to store blocks having a multiple-entry, single exit architecture.

7. (Previously presented) The front-end system of claim 4, wherein the block cache is to store complex blocks having multiple independent prefixes and a common suffix.

8. Canceled.

9. (Original) A method of managing extended blocks, comprising:

predicting an address of a terminal instruction of an extended block to be used,

determining whether the predicted address matches an address of a terminal instruction of a previously created extended block, and

selecting one of the extended block in the event of a match.

10. (Original) The method of claim 9, further comprising creating a new extended block when there is no match.

11. (Original) The method of claim 10, wherein the creating comprises:

receiving new instructions until a terminal condition occurs,

assembling the new instructions into an extended block,

determining whether an address of a terminal instruction in the new block matches an address of a terminal instruction of a pre-existing block, and

unless a match occurs, storing the new block in a memory.

12. (Original) The method of claim 11, wherein the storing comprises, when an older block causes a match, storing the new block over the old block in a memory if the old block is subsumed within the new block.

13. (Original) The method of claim 11, wherein the storing comprises, when an older block causes a match, dropping the new block if the new block is subsumed within the older block.

14. (Original) The method of claim 11, wherein the storing comprises, when an older block causes a match, creating a complex block if the new block and the older block share a common suffix but include different prefixes.

15. (Original) The method of claim 9, further comprising outputting instructions of the selected block for execution.

16. (Previously presented) A processing engine, comprising:

a front end stage to store blocks of instructions in a multiple-entry, single exit architecture when considered according to program flow, and

an execution unit in communication with the front end stage.

17. (Original) The processing engine of claim 16, wherein the front-end stage comprises:

an instruction cache system,

an extended block cache system, comprising:

a fill unit provided in communication with the instruction cache system,

a block cache, and

a selector coupled to the output of the instruction cache system and to an output of the block cache.

18. (Original) The processing engine of claim 17, wherein the block cache is to store the multiple-entry, single exit traces.

19. (Original) The processing engine of claim 17, wherein the extended block cache system further comprises a block predictor provided in communication with the fill unit and the block cache.

20. (Currently amended) A memory entry storing ~~A trace, comprising~~ a sequence of program instructions ~~stored together~~ as a trace ~~contiguous unit~~, the instructions defining a program flow that progresses from any instruction therein to a last instruction in the memory entry ~~trace~~ and in which the trace has multiple separate prefixes.

21. Canceled.

22. (Currently amended) The trace of claim 20, wherein the memory entry ~~trace~~ is indexed by an address of a terminal instruction therein.

23. (Previously presented) A memory comprising storage for a plurality of traces and means for indexing the traces by an address of a last instruction therein according to program flow.

24. (Previously presented) The memory of claim 23, wherein the traces include a plurality of instructions assembled according to program flow.

25. (Previously presented) The memory of claim 24, wherein the traces have a multiple entry, single exit architecture.

26. (Previously presented) The memory of claim 24, wherein at least one trace has separate prefixes and a common suffix, when considered according to program flow.

27. (Previously presented) The memory of claim 24, wherein at least one trace includes at least three segments of executable instructions in which, when considered according to program flow, first and second segments are mutually exclusive of each other and lead into the third segment.